Amendments to the Claims:

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

Claims 1-2 (canceled)

3. (currently amended): A method for loop-free multipath routing of data in a network, comprising:

(a) maintaining at each node i in a network,

(i) a main distance table (MDT) containing D_j^i and p_j^i, where D_j^i is the distance of node i to destination j and p_j^i is the predecessor to destination j on the shortest path from i to j,

(ii) said MDT further containing, for each destination j, successor set S_j^i, feasible distance FD_j^i, reported distance RD_j^i, and flags designated as $changed$ and $report-it$,

(iii) a main link table (MLT) T^i which is the node's view of the network and contains links represented by (m, n, d) where (m, n) is a link with cost d,

(iv) a neighbor distance table (NDT) for neighbor k containing D_{jk}^i and p_{jk}^i where D_{jk}^i is the distance of neighbor k to j as communicated by k and p_{jk}^i is the predecessor to j on the shortest path from k to j as notified by k,

(v) a neighbor link table (NLT) T_k^i which is the view that neighbor k has of the network as known to i and contains link information derived from the distance and predecessor information in the NDT, and

(vi) an adjacent link table (ALT) containing the cost l_k^i of an adjacent link to each neighbor k, wherein said cost is infinity if a link is down; and

(b) in response to receipt of an update message M from a neighbor k, detection of a change in cost of an adjacent link to k, or detection of a change in status of an adjacent link to k,

(i) updating the NDT and NLT for neighbor k with links (m, n, d) where $d = D_{nk}^i - D_{mk}^i$ and $m = p_{nk}^i$, and

(ii) constructing an MLT for neighbor k by merging topologies T_k^i and adjacent links l_k^i;

wherein a loop-free route for transmitting the data through the network is determined.

4. (original): A method as recited in claim 3, further comprising:

(c) for each destination j marked as *changed*, adding an update entry $\left[j, D_j^i, p_j^i\right]$ to a new message M'; and

(d) within a finite amount of time, sending message M' to each neighbor k.

5. (original): A method as recited in claim 4, wherein an update message comprises:

at least one update entry;

said update entry comprising a triplet $[j,d,p]$, where d is the distance of the node sending the message to destination j and p is the predecessor on the path to j; and

flags for synchronization designated as *query* and *reply*.

6. (original): A method as recited in claim 3, further comprising:

updating the MLT with link information reported by the neighbor k that offers the shortest distance from the node i to the head node m of the link if two or more NLTs contain conflicting information of link (m, n).

7. (original): A method as recited in claim 3, further comprising:

after merging topologies T_k^i and adjacent links l_k^i, running Dijkstra's shortest path algorithm to find the shortest path tree and deletes all links from the MLT that are not in the tree.

8. (original): A method as recited in claim 7, further comprising:

obtaining distances D_j^i and predecessors p_j^i from the MLT.

9. (original): A method as recited in claim 8, further comprising:

comparing the tree with the previous shortest path tree and reporting only the differences to the neighbors.

10. (original): A method as recited in claim 3, further comprising:

computing a successor set S_j^i by, for each destination j, allowing a node i to choose a successor having a distance to j as known to i that is less than the distance of node i to destination j that is known to a neighbor of node i.

11. (original): A method as recited in claim 10, further comprising synchronizing the exchange of update messages among neighbors using *query* and *reply* flags contained within the messages.

12. (original): A method as recited in claim 11, wherein if a node sends a message with a *query* flag set, said node must wait until a *reply* is received from all of said node's neighbors before said node is allowed to send the next update message.

13. (original): A method as recited in claim 12, wherein said node is deemed to be in an ACTIVE state when said node sends a message with a *query* flag set.

14. (original): A method as recited in claim 13, wherein said node is deemed to be in a PASSIVE state when said node has no message with a *query* flag set that is in transit or pending to be processed.

15. (original): A method as recited in claim 14:

wherein if a node in a PASSIVE state receives an event resulting in changes in its distances to a destination, before the node sends an update message to report a new distance, said node checks if the distance D_j^i to any destination j has increased above the previously reported distance RD_j^i; and

wherein if no distance has increased, then said node remains in a PASSIVE state;

wherein if a distance has increased, said node sets the *query* flag in the update message, sends said message, and goes into an ACTIVE state.

16. (previously presented): A method as recited in claim 13:

wherein a node in an ACTIVE state cannot send any update messages or add neighbors to any successor set;

wherein after receiving replies from all its neighbors, the node is allowed to modify the successor sets and report any changes that may have occurred since the time it has transitioned to ACTIVE state; and

wherein if none of the distances increased beyond the reported distance, the node transitions to PASSIVE state.

17. (original): A method as recited in claim 14:

wherein if a node receives a message with the *query* flag set when in PASSIVE state, said node modifies said node's tables and sends back an update message with the *reply* flag set; and

wherein if a node receives a message with the *query* flag set when in ACTIVE stage, said node modifies said node's tables and sends back an empty message with no updates and with the *reply* flag set.

18. (currently amended): A method for loop-free multipath routing of data in a network, comprising:

(a) maintaining at each node i in a network,

(i) a main distance table (MDT) containing D_j^i and p_j^i, where D_j^i is the distance of node i to destination j and p_j^i is the predecessor to destination j on the shortest path from i to j,

(ii) said MDT further containing, for each destination j, successor set S_j^i, feasible distance FD_j^i, reported distance RD_j^i, and flags designated as *changed* and $report-it$,

(iii) a main link table (MLT) T^i which is the node's view of the network and contains links represented by (m, n, d) where (m, n) is a link with cost d,

(iv) a neighbor distance table (NDT) for neighbor k containing D_{jk}^i and p_{jk}^i where D_{jk}^i is the distance of neighbor k to j as communicated by k and p_{jk}^i is the predecessor to j on the shortest path from k to j as notified by k,

(v) a neighbor link table (NLT) T_k^i which is the view that neighbor k has of the network as known to i and contains link information derived from the distance and predecessor information in the NDT, and

(vi) an adjacent link table (ALT) containing the cost l_k^i of an adjacent link to each neighbor k, wherein said cost is infinity if a link is down;

(b) in response to receipt of an update message M from a neighbor k, detection of a change in cost of an adjacent link to k, or detection of a change in status of an adjacent link to k,

(i) updating the NDT and NLT for neighbor k with links (m, n, d) where $d = D_{nk}^i - D_{mk}^i$ and $m = p_{nk}^i$, and

(ii) constructing an MLT for neighbor k by merging topologies T_k^i and adjacent links l_k^i;

(c) for each destination j marked as *changed*, adding an update entry $\left[j, D_j^i, p_j^i\right]$ to a new message M'; and

(d) within a finite amount of time, sending message M' to each neighbor k;

wherein a loop-free route for transmitting the data through the network is determined.

19. (original): A method as recited in claim 18, wherein an update message comprises:

at least one update entry;

said update entry comprising a triplet $[j,d,p]$, where d is the distance of the node sending the message to destination j and p is the predecessor on the path to j; and

flags for synchronization designated as *query* and *reply*.

20. (original): A method as recited in claim 18, further comprising:

updating the MLT with link information reported by the neighbor k that offers the shortest distance from the node i to the head node m of the link if two or more neighbor link tables contain conflicting information of link (m, n).

21. (original): A method as recited in claim 18, further comprising:

after merging topologies T_k^i and adjacent links l_k^i, running Dijkstra's shortest path algorithm to find the shortest path tree and deletes all links from the MLT that are not in the tree.

22. (original): A method as recited in claim 21, further comprising:

obtaining distances D_j^i and predecessors p_j^i from the MLT.

23. (original): A method as recited in claim 22, further comprising:

comparing the tree with the previous shortest path tree and reporting only the differences to the neighbors.

24. (currently amended): A method for loop-free multipath routing of data in a network, comprising:

(a) maintaining at each node i in a network,

(i) a main distance table (MDT) containing D_j^i and p_j^i, where D_j^i is the distance of node i to destination j and p_j^i is the predecessor to destination j on the shortest path from i to j,

(ii) said MDT further containing, for each destination j, successor set S_j^i, feasible distance FD_j^i, reported distance RD_j^i, and flags designated as *changed* and $report-it$,

(iii) a main link table (MLT) T^i which is the node's view of the network and contains links represented by (m, n, d) where (m, n) is a link with cost d,

(iv) a neighbor distance table (NDT) for neighbor k containing D_{jk}^i and p_{jk}^i where D_{jk}^i is the distance of neighbor k to j as communicated by k and p_{jk}^i is the predecessor to j on the shortest path from k to j as notified by k,

(v) a neighbor link table (NLT) T_k^i which is the view that neighbor k has of the network as known to i and contains link information derived from the distance and predecessor information in the NDT, and

(vi) an adjacent link table (ALT) containing the cost l_k^i of an adjacent link to each neighbor k, wherein said cost is infinity if a link is down;

(b) in response to receipt of an update message M from a neighbor k, detection of a change in cost of an adjacent link to k, or detection of a change in status of an adjacent link to k,

(i) updating the NDT and NLT for neighbor k with links (m, n, d) where $d = D_{nk}^i - D_{mk}^i$ and $m = p_{nk}^i$,

(ii) constructing an MLT for neighbor k by merging topologies T_k^i and adjacent links l_k^i; and

(c) computing a successor set S_j^i by, for each destination j, allowing a node i to choose a successor having a distance to j as known to i that is less than the distance of node i to destination j that is known to a neighbor of node i;

wherein a loop-free route for transmitting the data through the network is determined.

25. (original): A method as recited in claim 24, further comprising:

updating the MLT with link information reported by the neighbor k that offers the shortest distance from the node i to the head node m of the link if two or more NLTs contain conflicting information of link (m, n).

26. (original): A method as recited in claim 24, further comprising:

after merging topologies T_k^i and adjacent links l_k^i, running Dijkstra's shortest path algorithm to find the shortest path tree and deletes all links from the MLT that are not in the tree.

27. (original): A method as recited in claim 26, further comprising:

obtaining distances D_j^i and predecessors p_j^i from the MLT.

28. (original): A method as recited in claim 27, further comprising:

comparing the tree with the previous shortest path tree and reporting only the differences to the neighbors.

29. (original): A method as recited in claim 24, further comprising synchronizing the exchange of update messages among neighbors using *query* and *reply* flags contained within the messages.

30. (original): A method as recited in claim 29, wherein if a node sends a message with a *query* flag set, said node must wait until a *reply* is received from all of said node's neighbors before said node is allowed to send the next update message.

31. (original): A method as recited in claim 30, wherein said node is deemed to be in an ACTIVE state when said node sends a message with a *query* flag set.

32. (original): A method as recited in claim 31, wherein said node is deemed to be in a PASSIVE state when said node has no message with a *query* flag set that is in transit or pending to be processed.

33. (original): A method as recited in claim 32:

wherein if a node in a PASSIVE state receives an event resulting in changes in its distances to a destination, before the node sends an update message to report a new distance, said node checks if the distance D_j^i to any destination j has increased above the previously reported distance RD_j^i; and

wherein if no distance has increased, then said node remains in a PASSIVE state;

wherein if a distance has increased, said node sets the *query* flag in the update message, sends said message, and goes into an ACTIVE state.

34. (original): A method as recited in claim 31, wherein a node in an ACTIVE cannot send any update messages or add neighbors to any successor set.

35. (original): A method as recited in claim 31:

wherein after receiving replies from all its neighbors, the node is allowed to modify the successor sets and report any changes that may have occurred since the time it has transitioned to ACTIVE state; and

wherein if none of the distances increased beyond the reported distance, the node transitions to PASSIVE state.

36. (original): A method as recited in claim 32:

wherein if a node receives a message with the *query* flag set when in PASSIVE state, said node modifies said node's tables and sends back an update message with the *reply* flag set; and

wherein if a node receives a message with the *query* flag set when in ACTIVE stage, said node modifies said node's tables and sends back an empty message with no updates and with the *reply* flag set.

37. (currently amended): A method for loop-free multipath routing of data in a network, comprising:

(a) maintaining at each node i in a network,

(i) a main distance table (MDT) containing D_j^i and p_j^i, where D_j^i is the distance of node i to destination j and p_j^i is the predecessor to destination j on the shortest path from i to j,

(ii) said MDT further containing, for each destination j, successor set S_j^i, feasible distance FD_j^i, reported distance RD_j^i, and flags designated as *changed* and $report-it$,

(iii) a main link table (MLT) T^i which is the node's view of the network and contains links represented by (m, n, d) where (m, n) is a link with cost d,

(iv) a neighbor distance table (NDT) for neighbor k containing D_{jk}^i and p_{jk}^i where D_{jk}^i is the distance of neighbor k to j as communicated by k and p_{jk}^i is the predecessor to j on the shortest path from k to j as notified by k,

(v) a neighbor link table (NLT) T_k^i which is the view that neighbor k has of the network as known to i and contains link information derived from the distance and predecessor information in the NDT, and

(vi) an adjacent link table (ALT) containing the cost l_k^i of an adjacent link to each neighbor k, wherein said cost is infinity if a link is down;

(b) in response to receipt of an update message M from a neighbor k, detection of a change in cost of an adjacent link to k, or detection of a change in status of an adjacent link to k,

(i) updating the NDT and NLT for neighbor k with links (m, n, d) where $d = D_{nk}^i - D_{mk}^i$ and $m = p_{nk}^i$,

(ii) constructing an MLT for neighbor k by merging topologies T_k^i and adjacent links l_k^i, and

(iii) running Dijkstra's shortest path algorithm to find the shortest path tree and deletes all links from the MLT that are not in the tree; and

(c) computing a successor set S_j^i by, for each destination j, allowing a node i to choose a successor having a distance to j as known to i that is less than the distance of node i to destination j that is known to a neighbor of node i;

wherein a loop-free route for transmitting the data through the network is determined.

38. (original): A method as recited in claim 37, further comprising:

updating the MLT with link information reported by the neighbor k that offers the shortest distance from the node i to the head node m of the link if two or more NLTs contain conflicting information of link (m, n).

39. (original): A method as recited in claim 38, further comprising:

obtaining distances D_j^i and predecessors p_j^i from the MLT.

40. (original): A method as recited in claim 39, further comprising:

comparing the tree with the previous shortest path tree and reporting only the differences to the neighbors.

41. (original): A method as recited in claim 37, further comprising synchronizing the exchange of update messages among neighbors using *query* and *reply* flags contained within the messages.

42. (original): A method as recited in claim 41, wherein if a node sends a message with a *query* flag set, said node must wait until a *reply* is received from all of said node's neighbors before said node is allowed to send the next update message.

43. (original): A method as recited in claim 42, wherein said node is deemed to be in an ACTIVE state when said node sends a message with a *query* flag set.

44. (original): A method as recited in claim 43, wherein said node is deemed to be in a PASSIVE state when said node has no message with a *query* flag set that is in transit or pending to be processed.

45. (original): A method as recited in claim 44:

wherein if a node in a PASSIVE state receives an event resulting in changes in its distances to a destination, before the node sends an update message to report a

new distance, said node checks if the distance D_j^i to any destination j has increased above the previously reported distance RD_j^i; and

wherein if no distance has increased, then said node remains in a PASSIVE state;

wherein if a distance has increased, said node sets the *query* flag in the update message, sends said message, and goes into an ACTIVE state.

46. (original): A method as recited in claim 43, wherein a node in an ACTIVE cannot send any update messages or add neighbors to any successor set.

47. (original): A method as recited in claim 43:

wherein after receiving replies from all its neighbors, the node is allowed to modify the successor sets and report any changes that may have occurred since the time it has transitioned to ACTIVE state; and

wherein if none of the distances increased beyond the reported distance, the node transitions to PASSIVE state.

48. (original): A method as recited in claim 44:

wherein if a node receives a message with the *query* flag set when in PASSIVE state, said node modifies said node's tables and sends back an update message with the *reply* flag set; and

wherein if a node receives a message with the *query* flag set when in ACTIVE stage, said node modifies said node's tables and sends back an empty message with no updates and with the *reply* flag set.

49. (currently amended): A method for loop-free multipath routing of data in a network, comprising:

(a) maintaining at each node i in a network,

(i) a main distance table (MDT) containing D_j^i and p_j^i, where D_j^i is the distance of node i to destination j and p_j^i is the predecessor to destination j on the shortest path from i to j,

(ii) said MDT further containing, for each destination j, successor set S_j^i, feasible distance FD_j^i, reported distance RD_j^i, and flags designated as $changed$ and $report-it$,

(iii) a main link table (MLT) T^i which is the node's view of the network and contains links represented by (m, n, d) where (m, n) is a link with cost d,

(iv) a neighbor distance table (NDT) for neighbor k containing D_{jk}^i and p_{jk}^i where D_{jk}^i is the distance of neighbor k to j as communicated by k and p_{jk}^i is the predecessor to j on the shortest path from k to j as notified by k,

(v) a neighbor link table (NLT) T_k^i which is the view that neighbor k has of the network as known to i and contains link information derived from the distance and predecessor information in the NDT, and

(vi) an adjacent link table (ALT) containing the cost l_k^i of an adjacent link to each neighbor k, wherein said cost is infinity if a link is down;

(b) in response to receipt of an update message M from a neighbor k, detection of a change in cost of an adjacent link to k, or detection of a change in status of an adjacent link to k,

(i) updating the NDT and NLT for neighbor k with links (m, n, d) where $d = D_{nk}^i - D_{mk}^i$ and $m = p_{nk}^i$,

(ii) constructing an MLT for neighbor k by merging topologies T_k^i and adjacent links l_k^i;

(c) computing a successor set S_j^i by, for each destination j, allowing a node i to choose a successor having a distance to j as known to i that is less than the distance of node i to destination j that is known to a neighbor of node i; and

(d) synchronizing the exchange of update messages among neighbors using *query* and *reply* flags contained within the messages;

wherein a loop-free route for transmitting the data through the network is determined.

50. (original): A method as recited in claim 49, further comprising:

updating the MLT with link information reported by the neighbor k that offers the shortest distance from the node i to the head node m of the link if two or more NLTs contain conflicting information of link (m, n).

51. (original): A method as recited in claim 49, further comprising:

after merging topologies T_k^i and adjacent links l_k^i, running Dijkstra's shortest path algorithm to find the shortest path tree and deletes all links from the MLT that are not in the tree.

52. (original): A method as recited in claim 51, further comprising:

obtaining distances D_j^i and predecessors p_j^i from the MLT.

53. (original): A method as recited in claim 52, further comprising:

comparing the tree with the previous shortest path tree and reporting only the differences to the neighbors.

54. (original): A method as recited in claim 49, wherein if a node sends a message with a *query* flag set, said node must wait until a *reply* is received from all of said node's neighbors before said node is allowed to send the next update message.

55. (original): A method as recited in claim 54, wherein said node is deemed to be in an ACTIVE state when said node sends a message with a *query* flag set.

56. (original): A method as recited in claim 55, wherein said node is deemed to be in a PASSIVE state when said node has no message with a *query* flag set that is in transit or pending to be processed.

57. (original): A method as recited in claim 56:

wherein if a node in a PASSIVE state receives an event resulting in changes in its distances to a destination, before the node sends an update message to report a new distance, said node checks if the distance D_j^i to any destination j has increased above the previously reported distance RD_j^i; and

wherein if no distance has increased, then said node remains in a PASSIVE state;

wherein if a distance has increased, said node sets the *query* flag in the update message, sends said message, and goes into an ACTIVE state.

58. (original): A method as recited in claim 55, wherein a node in an ACTIVE cannot send any update messages or add neighbors to any successor set.

59. (original): A method as recited in claim 55:

wherein after receiving replies from all its neighbors, the node is allowed to modify the successor sets and report any changes that may have occurred since the time it has transitioned to ACTIVE state; and

wherein if none of the distances increased beyond the reported distance, the node transitions to PASSIVE state.

60. (original): A method as recited in claim 56:

wherein if a node receives a message with the *query* flag set when in PASSIVE state, said node modifies said node's tables and sends back an update message with the *reply* flag set; and

wherein if a node receives a message with the *query* flag set when in ACTIVE stage, said node modifies said node's tables and sends back an empty message with no updates and with the *reply* flag set.